

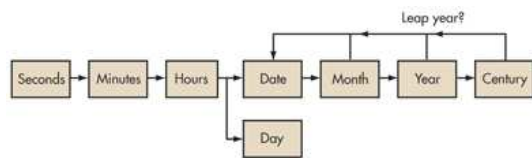
Real-Time Clock Programming Requires Caveats

0

Larry A. Jones
July 31, 2012

When developing code to operate a real-time clock (RTC), it is often beneficial to understand how the clock core operating logic has been defined. We all have a fundamental comprehension of how a clock and calendar should operate when properly programmed, but insight into the expected chip behavior if improper or illegal contents are somehow installed also is helpful. The software must ensure the proper values are written to the chip.

RTC counter chains can be found in many BCD-formatted (binary-coded decimal) RTCs (*Fig. 1*). Each counter register has defined minimum and maximum values, and most are preset to their appropriate minimums on initial power application.



1. Basic RTC counter chains can be found in many BCD-formatted RTCs.

As a natural function of timekeeping, when the seconds count reaches the maximum value, the next increment causes a carry to the minutes register and the seconds rolls over to the minimum value. A minutes rollover carries to the hours. The hours rollover carries to both the date and arbitrary day-of-week registers. The date rollover carries to the month register. The month rollover carries to the year register.

If applicable, the year rollover carries to the century register. Some variations on the counter implementation may include a “hundredths of a second” register (precedes seconds) and a century bit to flag the year register rollover (*Table 1*).

TABLE 1: MAXIMUM AND MINIMUM VALUES OF EACH REGISTER			
Register	Minimum (hex)	Maximum (hex)	Carry to register
Hundredths	0	99	Seconds
Seconds	0	59	Minutes
Minutes	0	59	Hours
Hours (12-hour mode)	41	72	AM → PM, PM → AM + day and date
Hours (24-hour mode)	0	23	Day and date
Day	1	7	—
Date	1	31*	Month (*month and year dependent)
Month	1	12	Year
Year	0	99	Century (register or bit)
Century	0	99	—

Register Descriptions

For each register, the normal state machine logic is to increment the BCD count from the minimum to the defined maximum and then roll back over to the minimum while applying the carry to the next register. For an example, the date register BCD count sequence for a 30-day month would be 01h...09h, 10h...19h, 20h...29h, 30h, and then roll back to 01h. The ramifications of loading improper values are based upon the specific component and which register(s) was actually involved. Illogical time and date entries may result in unexpected operation.

The hundredths of a second register (if applicable) counts from 00h to 99h. The next increment causes the counter to roll over to 00h, and the carry is applied to the seconds register. All 8 bits in this register are read/write capable. Devices containing this register are internally clocked from the 4-kHz time base (32.768 kHz/8).

The seconds register counts from 00h to 59h. The next increment causes the counter to roll over to 00h, and the carry is applied to the minutes register. Only the least significant 7 bits in this register are usually read/write capable. Most devices that don’t have a hundredths of a second register are internally clocked from the 1-Hz time base.

The minutes register also counts from 00h to 59h. The next increment causes the counter to roll over to 00h, and the carry is applied to the hours register. Only the least significant 7 bits in this register are read/write capable.

The hours register accommodates either a 12-hour format (with an AM/PM designator) or a 24-hour clock. A control bit (usually bit6) is used to identify the clock operating mode. Only the least significant 7 bits in the hours register are read/write capable. In the 24-hour mode (bit6 = 0), the hours register counts from 00h to 23h. The next increment causes the counter to roll over to 00h, and the carry is applied to both the date and day registers.

In the 12-hour mode (bit6 = 1), the hours register count sequence, starting at "12AM," counts 52h, 41h...49h, 50h, and then 51h. The next increment causes the counter to roll over to 72h with the carry applied to the /AM//PM bit (PM = 1). The register then counts 72h, 61h...69h, 70h, 71h. The next increment causes the counter to roll over to 52h as PM is cleared, with the carry applied to both the date and day registers. For clarification, midnight is 12:00:00 AM and noon is 12:00:00 PM.

The day (day-of-week) register is the most simplistic counter, with an intended content of 01h to 07h. The next increment causes the counter to roll over to 01h with no carry. Should a user accidentally deposit a 00h content into this register, there is no adverse effect upon the other timekeeping registers, but the first week of clock operation will have eight days (0 → 7).

The day register only has three functioning bits. So even though it initially seems impossible for the user to load a value greater than 7, any hex value greater than 7 written would store the bad value logic-OR'd with the three functioning bits (07h) (e.g., writing 1Fh would store a 07h, or writing F5h would store a 05h). Normal calendar convention is to define Sunday = 1, but that definition is arbitrary and fully under the user's control.

The date register counts from 01h to the defined end of the present month. The end of the month depends upon the present month and year content (in the case of February). Only the least significant 6 bits in the date register are read/write capable (Table 2).

TABLE 2: END DATES FOR EACH MONTH		
Calendar month	Numeric month	End date
January, March, May, July, August, October, December	1, 3, 5, 7, 8, 10, 12	31
April, June, September, November	4, 6, 9, 11	30
February	2	28 (typical year), 29 (leap year)

The month register counts from 01h to 12h. The next increment causes the counter to roll over to 01h, and the carry is applied to the year register. To address Y2K issues in the late 20th century, a century bit was included in the most significant bit (MSB) of the month register on components that did not include a full 8-bit century register.

This century bit facilitated easy detection of the century change, as the bit changes state upon a year register rollover from 99h to 00h. Care should be taken when loading new calendar values to avoid corrupting the century bit logic condition. Please refer to the register map for the specific component in question. Conventional register placement leaves bits 7 and 4:0 as read/write capable.

The year register counts from 00h to 99h. The next increment causes the counter to roll over to 00h. The carry is applied to either the century register or the century bit, whichever is applicable to the component in question. All 8 bits in this register are read/write capable.

Leap year compensation is based upon the era of the specific chip design. Applicability is normally annotated on the cover page of that datasheet. For most chip designs that do not include a century register, the leap year compensation functions properly through 2099 and is assumed to be the result of:

If $\text{MODULO}(\text{CALENDAR_YEAR}/4) == 0$

Due to the era of existing chip designs, many of today's RTC components with a century register apply the same algorithm solely upon on the content of the year register. Therefore, they will incorrectly designate 2100 as a leap year. The Maxim DS1347 RTC utilizes the full Gregorian algorithm to determine leap year adjustments for February (Fig. 2). The century register (if applicable) counts from 00h to 99h. All 8 bits in this register are read/write capable.

```

Leap_Year = 0 ; default to No
If MODULO (CALENDAR_YEAR/4) == 0 ; divisible by 4?
    If MODULO (CALENDAR_YEAR/100) == 0 ; divisible by 100?
        If MODULO (CALENDAR_YEAR/400) == 0 ; divisible by 400?
            Leap_Year = 1
        end ; if
    else ; if
        Leap_Year = 1
    end ; if+
end ; if

```

2. Last date of month is affected by the month as well as the leap year status.

Results When Illegal Data Is Written

Up to this point, we have discussed the proper counting methods used by the RTC, given the legal range of values for the register in question. Occasionally a question may arise concerning expected counter behavior when improper values were initially installed. The answer is deterministic and only requires a copy of the component's register map to arrive at a conclusion.

To access the result of loading an improper value, perform a logic AND of the written value with the write-enabled bits in the register in question. If the resulting content is less than the register's maximum counter value, the register should subsequently increment from that AND-ed value. Rollover will occur when the bit states in the register eventually coincide with the expected maximum value for that counter. The following examples highlight this effect.

Example 1: Invalid Month

The month register of a DS1337 serial RTC chip has a valid range from 1 to 12. The most significant bit is the century bit. The BCD month portion is 5 bits.

Writing a value of 2Ah to the register will result in a value of 0Ah being stored in the register. This is an invalid BCD value. If the desired value was month 10 then the value should have been 30h, resulting in a stored value of 10h. Bit 6 is also set for both 2Ah and 30h. This sets the chip into 12-hour mode.

Example 2: Invalid Hour

The hours register range of a DS1390 low-voltage RTC is based on whether the chip is running in 12- or 24-hour mode, but there are values that are invalid regardless of the mode. The hours register has a write mask of 7Fh. Writing a value of FFh will result in a stored value of 7Fh. This is invalid for both modes.

Example 3: Invalid Hour Depending Upon Mode

Writing a 38h into the hours register of a DS1341 low-current RTC results in the unchanged value being stored in the register since the 7Fh mask is used regardless of mode. Now it turns out that this value is invalid for both modes because it is outside of the range noted in Table 1. It exceeds the 24-hour limit of 23h and is below the 41h for 12-hour mode.

Example 4: Invalid Seconds

Writing AAh into the seconds register of a DS1302 trickle-charge timekeeping chip causes a problem because this chip uses bit 7 for a clock halt function. This stops the clock so no time keeping occurs until the bit is set to 0. In this case, the clock will be stopped and the value of 2Ah will be saved in the register. The actual value is really irrelevant since the clock will not start counting until bit 7 is reset, and this requires writing a new value into the register.

It is still possible to write an invalid seconds value while allowing the chip to handle its timekeeping duties. For example, a value of 60h is a problem since it exceeds the register's limit of 59h or 59 minutes.

Example 5: Invalid Date

Writing 2Dh into the date register of a DS1338 serial RTC is problematic. The 6-bit mask (3Fh) for the register will allow the value to be stored but it exceeds the limit of 31h, although this limit is month specific. The upper limit ranges from 28h to 31h depending upon the month and leap year.

Conclusion

By using published register information and the deterministic nature of the state machine logic, some behavioral traits resulting from errant programming can be recognized during the system debugging phase. This symptom recognition can reduce a product's time-to-market.

Tags: real-time clock serial RTC chip

Like

1

Tweet

Share

 **Subscribe to the Electronic Design eNewsletters**

* Email Address:

* Country:

*** Select your eNewsletter options below:**

☐ ED Update - Industry-wide News (weekly)

☐ Embedded Update - Industry Specific News (weekly)

☐ Power & Analog Update - Industry Specific News (weekly)

☐ Engineering TV - Engineering Video's (daily)

☐ Products of the Week - New Product Updates (weekly)

☐ Europe Newslne - European Industry News (monthly)

Subscribe